Vertex Clustering of Augmented Graph Streams

Ryan McConville* Weiru Liu* Paul Miller*

Abstract

In this paper we propose a graph stream clustering algorithm with a unified similarity measure on both structural and attribute properties of vertices, with each attribute being treated as a vertex. Unlike others, our approach does not require an input parameter for the number of clusters, instead, it dynamically creates new sketch-based clusters and periodically merges existing similar clusters. Experiments on two publicly available datasets reveal the advantages of our approach in detecting vertex clusters in the graph stream. We provide a detailed investigation into how parameters affect the algorithm performance. We also provide a quantitative evaluation and comparison with a well-known offline community detection algorithm which shows that our streaming algorithm can achieve comparable or better average cluster purity.

1 Introduction

Online streaming data analytics is an emerging research area. Much of the data generated can naturally be modelled as graphs, such as from sensor and social networks. These streams constantly evolve, producing dynamic graphs of rich information. Graph stream clustering algorithms aim to extract useful information online from these graphs. Furthermore, these networks tend to be extremely large which introduces computation and memory demands, adding further challenges to the graph stream clustering problem.

Clustering is a widely used approach to analysing data[10]. Its objective is to identify sets of items where objects in the same set show high levels of similarity to each other. In the graph or network setting, this is often called community detection[9] and the objective is to identify sets of items that are densely connected with each other. Typically this is done in an offline, static environment, where the data has been completely collected. However, now research has increasingly focused on dynamic, online clustering where the data is clustered as it arrives[13].

Clustering streams of graph data is regarded as a challenging problem due to constraints such as the number of passes on the data, having a large stream of potentially unbounded length, as well as computational and memory demands. In graph stream clustering, the updates to the graph dataset arrive in a continuous way and can be considered as individual transaction graphs. These transaction graphs themselves are relatively small but form a part of the extremely large aggregated graph and are usually defined over a massive domain of distinct vertices such as users on a social network or hyperlinks on the world wide web. The size of the aggregated graph prevents it from being stored in memory and storage on disk would be too slow for an online setting. Furthermore, due to the large volume, each transaction graph can only be examined with few passes, typically one.

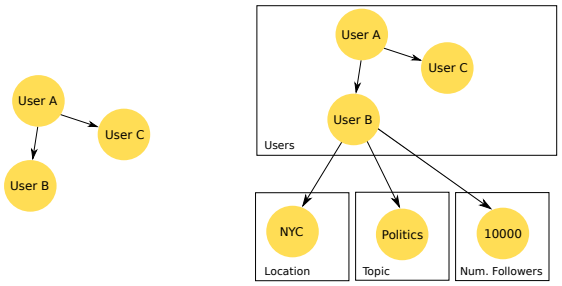


Figure 1: How graph augmentation enlarges the transaction graph G_t (left) to G_s (right). All of G_s (right) is the augmented subgraph.

Example 1: A social network consists of many people where directed messages are exchanged publicly between groups of users. Each new message arrives from the stream and is modelled as a graph consisting of directed edges from the user who sent the message to the users who received it. Let User-A be the sender and User-B and User-C be the receivers of one single message.

If applying clustering, or community detection, to this stream described in Example 1, one would aim to assign each user to the cluster that they are most similar to based on structural connectivity of their communication patterns. Furthermore, attributes of each vertex can be considered meaningful information and be taken into account during the clustering process, and they can be directly incorporated into the graphs via augmentation[22]. We illustrate how augmentation will affect our example graph in Figure 1. In this example, these attributes consist of a users profile

*Queen's University Belfast, {rmcconville07, w.liu, p.miller}@qub.ac.uk

on a social network but other examples include the title and citations of a scientific paper in a stream of paper publications, or the transaction history of a user during a purchase in a financial transaction stream. These attributes can lead to the discovery of clusters that may otherwise have been missed if only structural connectivity was considered. Ideally an algorithm should consider both structural and attribute information to achieve a better clustering result. For example, small sparsely connected communities may be completely unconnected when attributes are not considered, but otherwise, densely connected around attributes. Attributes are considered as vertices and this allows a unified similarity function to be applied to both the original vertices and the augmented vertices.

The constraints of the graph stream setting, such as the limited number of passes and a huge amount of data, make many existing substructural analysis algorithms, especially algorithms that compare graphs on a pairwise basis, unsuitable for processing streaming data.

To the best of our knowledge, vertex clustering has not been applied to augmented graph streams. *GSSClu*[21] studies a related problem of clustering the graph objects, rather than the vertices themselves. In many domains, such as Example 1, we may be more interested in clustering each vertex (user) of a new transaction graph individually rather than the transaction graph (communication) as a whole. In doing so, our approach looks at each individual vertex and its attributes as well as its relationship to the vertices in every transaction graph.

Therefore our approach will not only take into account edges, but *also* consider the vertices of each graph object individually, as well as the attributes as vertices. The original idea behind the sketches used in the current literature for cluster storage (see Section 4.1), lacks the ability to retrieve an arbitrary vertex without enumerating all combinations of vertices and edges. We overcome this weakness by storing each graph vertex, as well as edges, in the clusters. For instance in Example 1, if User-A communicates with another user, User-D, it would be beneficial to know to which cluster User-A belongs. Our approach supports this finer grained analysis.

Vertex clustering can be considered more challenging than graph object clustering. One challenge arises from the increased likelihood that the same vertex may appear multiple times in the stream, and potentially with different attribute values. Referring back to Example 1, User-A may have sent the message from a specific location, but this is an attribute that is likely to change. This problem is related to the phenomenon of concept drift[17].

Another challenge is to estimate the number of clusters a priori. Many graph clustering algorithms require an input parameter specifying the number of clusters to be created, which is a known limitation in clustering[11]. This problem can be even more challenging in the stream setting where the data stream may be unbounded and analysis is limited. Our algorithm does not require setting the number of clusters up front, instead, we dynamically create new clusters under certain conditions, outlined in Section 4.5. Our main contributions in this work are:

- The first augmented vertex graph stream clustering algorithm.
- A novel cluster definition that samples the best substructures and vertex label values.
- A novel graph stream cluster merging scheme.
- A quantitative evaluation of our technique and a comparison with the well-known Louvain[7] offline community detection algorithm.

The rest of this paper is organised as follows. Section 2 briefly surveys related work. In Section 3, we define how we augment the graph stream while discussing its benefits. In Section 4 we describe how we adapt the sketch for our purpose, how our clusters and likeness measures are defined and our algorithm in detail. We proceed in Section 5 to evaluate our approach on two public datasets. We conclude in Section 6.

2 Related Work

In the literature techniques exist for mining graphs in the traditional static context. Many of them focus on the problem of static vertex clustering on a single graph with the aim to group similar vertices together based on their linkage properties. One of the earliest approaches was graph partitioning[12] which partitioned weighted graphs with respect to edge weights. The Louvain method[7] is a widely used algorithm for the detection of communities in large networks. It uses the modularity metric which favours dense intra-cluster and sparse inter-cluster connections, but requires multiple passes over the entire dataset. Not limited to only looking at the structural relationships between graph vertices, [22, 23] integrated attributes into the clustering process by adding these attributes as vertices to the original graph via augmentation for an offline random walk based approach. In two other recent approaches, CESNA[19] integrated network connectivity with attribute similarity to detect overlapping communities and [18] developed a probabilistic Bayesian model for attributed graphs. This recent and ongoing interest

in the relationship between vertices and their attributes in clustering and community detection have provided many examples of the advantages. However, many of the proposed techniques in the literature, such as those mentioned, are not suitable for the streaming data setting as they require multiple passes over the data or rely on the ability to be able to retrieve previous data.

Recently in the graph stream setting, techniques have been proposed which address issues with mining graph streams. The focus of techniques range from outlier detection[5], classification[2], dense pattern mining[3], as well as clustering algorithms[4, 20]. Some research has been carried out into dynamic community detection[16, 15, 14] but none considered augmented graph streams and probabilistic data structures for 'big data'. The approaches only take the structural linkage information from the graph into account during the mining process. One exception, a very recent work, $GSSClu$[21] incorporated attributes into the clustering algorithm as what they called 'side information' for each graph object from the stream as a whole, rather than the individual vertices of the graph object.

We propose a novel approach to addressing the issues of one pass processing of graph stream data and association of attribute information to vertices in a graph. We incorporate attributes directly into the graph stream via graph augmentation, similar to what is proposed by [22] (but our approach is applied to graph streams) for the static domain. To the best of our knowledge, this is the first attempt to apply vertex clustering to the problem of streaming augmented graphs.

3 Graph Vertex Augmentation with Attributes

DEFINITION 1. *A graph $G = (V, E)$ consists of a set of vertices V and edges $E \subseteq V \times V$. Two vertices v,w are adjacent iff $\langle v, w\rangle \in E$. If the tuple $\langle v, w\rangle$ is ordered, the graph is directed, otherwise it is undirected.*

DEFINITION 2. *A graph $G_t = (V_t, E_t)$ is a subgraph of G, denoted $G_t \subseteq G$, iff $V_t \subseteq V, E_t \subseteq E \cap (V_t \times V_t)$.*

DEFINITION 3. *A graph stream $G = \{G_1, G_t, \ldots, G_n\}$ is a stream of subgraphs; For clarity, we call each G_t from the stream a transaction graph.*

The stream of transaction graphs vary by domains, such as graphs consisting of emails, telephone calls, URLs on webpages, financial transactions or communications between users on the Twitter social network. These transaction graphs can then be augmented with attributes of the vertex to provide more information and discover more meaningful clusters based on both structural and attribute similarity.

DEFINITION 4. *Let A be a set of discrete attribute labels $A = \{a_1, \ldots, a_n\}$ associated with a graph stream. Let $a \in A$ be an attribute label with the domain $D_a = \{d_1, \ldots, d_m\}$. The set of attribute-value pairs $\lambda = \{\langle a, d\rangle : a \in A, d \in D_a\}$.*

Each transaction graph G_t will be partitioned into $|V_t|$ augmented subgraphs. For each $v_t \in V_t$ we create one augmented subgraph that contains the set of attribute value pairs for vertex v_t in addition to original transaction graph G_t.

DEFINITION 5. *An augmented subgraph G_s of the graph G_t is denoted $G_s = (v_t, V_s, E_s, \lambda_s)$ where v_t is the structure vertex of G_s, V_s is the set of other structure vertices in G_t, E_s is the set of edges, and λ_s the set of attribute-value pairs belonging to v_t.*

For clarity, like [22], we will refer to the original vertices as *structure vertices* and the original edges as *structure edges*. Attribute vertices that were added via the augmentation process will be referred to as *attribute vertices* and the resulting edges from these vertices referred to as *attribute edges*.

When a vertex v_t has a value for attribute a_j, an attribute vertex for a_j is added to G_s, with an attribute edge between v_t and this new attribute vertex.

The set of discrete attribute labels A from Definition 4 plus the label for the structure vertex v_t (labels are also the names of the vertices) will be referred to as the *Complete Label Set* or CLS.

An alternative approach to augmentation may be to associate attributes as feature vectors for each vertex. However, this does not lend itself well to providing a unified similarity measure of attribute and structural information, as the attributes would be explicitly excluded from any similarity function that measures connectivity. As shown, we choose to add each attribute into the graph as a vertex to provide a unified similarity measurement framework for vertex clustering with augmented graph streams. In this generalised framework, each vertex label, including those of the attribute vertices, are treated as discrete labels. In our unified approach, by augmenting the graph with attributes as vertices, the same similarity measurement for structure vertices is applied to attribute vertices and the varying importance of each can be controlled by weights applied to each attribute.

We stress how our approach differs to that of $GSSClu$[21]. Their method of incorporating these attributes mean that each set of side information is associated with the graph object as a whole, which limits the effectiveness in domains where side information may be associated uniquely to individual vertices in each graph.

For example, if we are clustering graph objects consisting of user communication on a social network, user locations may be a useful attribute as part of the side information. However, vertices in the graph object may be associated with different locations. Since we are proposing vertex clustering, it is natural to only consider the attributes relevant to the vertex being clustered. Where as for $GSSClu$ each cluster is a set of transaction graphs (G_t), with our approach each cluster is a set of vertex augmented subgraphs (G_s).

4 Methodology

4.1 Sketch Statistics. In the graph stream setting, we require a method to overcome the computational and memory limitations that come with large amounts of data. Furthermore, due to our augmentation process and storage of vertices as well as edges, we expect the volume of data to be even greater. One approach to solve this may be to use a form of random sampling of the incoming data stream[5], however we would like to keep a count of each item we see in the stream, including sparse regions, which is easier if we can process all items. An approach that supports processing all items in the stream, but still accommodates large amounts of data is a sketching data structure called Count-Min[8] which maintains summary statistics over the entire data stream, storing a constant sized sub-linear space representation in memory. Sketch data structures such as Count-Min can be considered a 'synopsis' of the input data stream, maintaining specific summary statistics, providing approximations over accuracy for increased efficiency.

We adapt the Count-Min sketch for estimating frequency statistics of the vertices and edges in the graph stream. Count-Min is a sketch approach to summarising data streams and approximate aggregation. Each sketch table is maintained as a 2D array with $ln(1/\delta)$ rows (denoted d) and e/ϵ columns (donated w) where e is the base of the natural logarithm. δ and ϵ are input parameters required by Count-Min to determine the sketch dimensions. d different hash functions are generated randomly from a pairwise independent family, each of which corresponds to one of the 1D rows with w cells. For example, when a new vertex v is being added to the sketch, each hash function is applied to v and maps it to a hash value hv_j with range $[0, w-1]$. For the j^{th} hash function, the frequency of the vertex v is added to the ${hv_j}^{th}$ column on the j^{th} row of the sketch. To retrieve the estimated frequency of a vertex v we map it to d cells by applying d hash functions. The frequency of the vertex is the minimum value among all these d cells. This value will either be correct or an overestimation with maximum bound of $\epsilon \cdot T$ with probability at least $1-\delta$ for a data stream with T arrivals. The same process occurs for storing and retrieving edge frequencies. The edge is derived by concatenating the vertices of the edge; $e = \langle v_i, v_j \rangle$ is $v_i \oplus v_j$.

Our extension of the use of Count-Min in the graph stream setting is to not only estimate the frequency statistics of edges, but also of vertices.

4.2 Cluster Definition. As we aim to divide vertices into clusters, we need an efficient method to maintain the contents of each cluster.

$ESketch\,(c_l)$ for storing the frequency of each structure and attribute edge in c_l.

$VSketches\,(c_l) = \{sketch\,(l_0), \ldots, sketch\,(l_m)\}$ for vertices, one sketch for each label $l \in CLS$.

$SoB(c_l)$: A vector of frequencies of substructures (called Sample of Best) assigned to cluster c_l which are the top $|SoB\,(c_l)\,|$ frequencies. That is, $SoB(c_l) = \{s_1, s_2, \ldots, s_q\}$ where each s_c is the total frequency value of an augmented subgraphs vertices and edges in the cluster.

W: a set of weights, each of which is associated with a label $l \in CLS$ and represents the importance of this label in the clustering processing.

$HGS(c_l) = \{heap\,(l_0), \ldots, heap\,(l_m)\}$ maintains a set of heaps for each label $l \in CLS$. Each $heap(l_j)$ holds the top p most frequent values for label l_j in their non-hashed form. This is used by the cluster merging algorithm to be discussed in Section 4.6.

By including a sketch of vertices in each cluster, we can extend queries to include the frequency of vertices in each cluster, not just edges. By maintaining heaps of the most frequent values for each vertex, including attribute vertices, we retain the most frequent values for vertices which can be used during the merging of clusters, without enumerating all past inputs.

4.3 Graph Preprocessing. Since we are proposing a unified similarity measurement framework for augmented graphs, we consider the situation where each structure and attribute vertex label is discrete without losing generality. Discrete types are chosen as our similarity function is based on frequency statistics of vertex and edge labels. The vertex labels of the edge are ordered as stated in Definition 5. Ordering is important for consistent cluster sketch statistics as an edge $label_a - label_b$ will hash to a different value, and therefore different sketch locations, than $label_b - label_a$.

4.4 Likeness. For a structure vertex v_t in a transaction graph, the goal is to place its augmented subgraph $G_s = \{v_t, V_s, E_s, \lambda_s\}$ into the cluster that G_s is most similar to based on other items in the cluster. We use

K to denote a set of clusters.

To define a unified measure in this subsection, we use v_a to denote an attribute-value pair in λ_s from a given G_s. The frequency of a structure vertex or an attribute-value pair in a cluster c_l, denoted $F_V(v, c_l)$, (v is either v_t or v_a), is simply the number of times it has been placed into cluster c_l previously.

The membership of v in cluster c_l, denoted as $MS_V(v, c_l)$, is the frequency of v in c_l taking into account the weight of the vertex. The weights for attributes are predefined to reflect how important they are. We let the structure vertex weight be 1.

$$MS_V(v, c_l) = F_V(v, c_l)\, w(v) \tag{4.1}$$

For $\langle v_t, v_j \rangle$ in E_s the membership $MS_E(\langle v_t, v_j \rangle, c_l)$ in cluster c_l, is the frequency of $\langle v_t, v_j \rangle$ in cluster c_l, taking into account the weight of the vertex v_j from the edge. When v_j is a structure vertex we let the weight be 1, when it represents an attribute value pair it is the attribute weight.

$$MS_E(\langle v_t, v_j \rangle, c_l) = F_E(v_t \oplus v_j, c_l)\, w(v_j) \tag{4.2}$$

where $\oplus$ is the concatenation operator on the vertex labels strings v_t and v_j.

The likeness expectation of the current subgraph G_s to an existing cluster is calculated via the likeness as shown in formula 4.3.

$$likeness(G_s, c_l) = \frac{\left(\sum_{\forall v \in \{v_t\} \cup \lambda_s} (MS_V(v, c_l)) + \sum_{\forall \langle v_t, v_j \rangle \in Es} (MS_E(\langle v_t, v_j \rangle, c_l)) \right)}{mean(SoB(c_l))} \tag{4.3}$$

$max(likeness(G_s, c_l))$ is the most similar cluster to the current subgraph.

The definition of *likeness* between an augmented subgraph and a cluster is the central idea of our *unified similarity measurement framework.*

4.5 Conditions for Cluster Creation. If there is no single cluster c_l that G_s is most similar to, then we create a new cluster for it, that is when the maximum likeness value is 0 which occurs when subgraph G_s has no similarity to any current cluster.

The other condition for cluster creation is when G_s is an outlier in the cluster to which it was closest. We define an outlier range so we can detect and handle the situation that occurs when the most similar cluster for G_s is too dissimilar. To accomplish this we use the cluster SoB data structure and determine an outlier value to be greater than 3σ less than the current mean of the SoB samples.

4.6 Cluster Merging. As commonly used in the information retrieval domain for finding similarity between sets of documents, we use cosine similarity for measuring the similarity between sets of clusters. With the use of a probabilistic data structure such as Count-Min for storing cluster statistical contents, this would normally be infeasible in the stream setting as retrieving sketch hash values requires the sketch hash key. This would be impractical to store for large amounts of data. However, we maintain a set of heaps HGS containing the most frequent values of each label in CLS. With each HGS represented as vectors of feature values, the degree of similarity between two clusters is computed as the average cosine of the angle between them.

$$sim(HGS(c_l), HGS(c_u)) = \frac{(HGS(c_l) \cdot HGS(c_u)}{|(HGS(c_l)||HGS(c_u)|} \tag{4.4}$$

where $HGS(c_l)$ and $HGS(c_u)$ are the sets of heaps, one for each label in CLS, of most frequent label values. If the resulting average similarity value is greater than a predefined similarity threshold, then the two clusters are to be merged.

Count-Min sketches can be merged via entry-wise summation and the merged sketch becomes the union of the previous two sketches[1]. Each cluster SoB and HGS is also updated in this merging operation.

The cosine similarity is a useful metric for determining the similarity of our clusters as it is a measurement of orientation and not magnitude. This is useful as merging may occur between newly created clusters and older and larger cluster with a larger magnitude.

5 Experiments

5.1 Datasets. As the approach of vertex clustering in graph streams is novel and no baselines exist for comparison, one of our experiments will use the CORA dataset[1] which was used by a related approach[21] to clusters papers on their topics using a graph stream with associated side information. The CORA dataset consists of scientific articles in the computer science domain, each of which can be represented as a graph with co-author relationships as edges. We use the same attributes as [21], the paper terms and citations, and augment each author subgraph with them as defined in Definition 5. One unique challenge of our approach on a dataset such as CORA is that graph object clustering approaches, such as [21], would be clustering papers, which have a single associated topic, where as we are

[1] http://people.cs.umass.edu/~mccallum/data/cora-classify.tar.gz

Algorithm 1: Augmented Vertex Subgraph Clustering Algorithm (AVSCA)

```
Input: X: cluster merge smoothing parameter
foreach newly_received_graph G_t do
    Split G_t into augmented subgraphs G_s as defined in Definition 5;
    foreach subgraph G_s do
        for l = 1 to K do
            compute likeness(G_s, C_l) defined in equation 4.3
        let C_closest be the set of closest clusters;
        if |C_closest| >1 then
            Create new cluster with G_s.
        else if likeness(G_s, C_l) < Outlier_Range(C_closest) then
            Create new cluster with G_s;
            Report G_s as an outlier;
        else
            Update C_closest cluster by inserting the augmented subgraph G_s;
    if graph_count % X == 0 then
        call Cluster Merge Step();
    graph_count += 1;
```

Algorithm 2: Cluster Merge Step

```
Input: similarity_threshold: Threshold for merging similarity
Input: C: Set of clusters
Output: D Set of clusters
Let K be the |C|;
for l = 1 to K do
    for m = 1 to K do
        if m != l then
            Calculate cosine similarity of HGS(c_l) and HGS(c_m) as defined in equation 4.4;
Let similarity_list be the list of similar cluster pair tuples in descending order by similarity;
foreach similarity_pair sp in similarity_list do
    if similarity_score(sp) >similarity_threshold then
        Merge clusters in sp together;
        Remove any remaining similarity_pairs containing either cluster from sp;
```

clustering authors who may publish papers on different topics throughout the stream.

The Million Song Dataset (MSD)[6][2] is a 280GB dataset consisting of data from one million songs. The data associated with each song includes basic metadata as well as features extracted from the audio of each song. The last.fm dataset[3] is a collection of songs with associated 'tags' that have been provided by users of the last.fm service. The tags can be linked with songs from the MSD. These collaboratively labelled tags can be interpreted as the genres for the songs. By discovering songs that are similar to each other, recommendations of similar songs (i.e. those in the same cluster) can be provided to users based on the songs that they like. This dataset also highlights the importance of augmenting attributes as, unlike the CORA graph, songs will typically only appear once and lack any natural structural relationship to other songs outside of the album they are present on. Without attributes, it would be challenging to exploit the structural relationship between songs for clustering.

We first choose 10 of the most common tags from the dataset to use as genres. They are *rock, pop, blues, hip-hop, country, reggae, electronic, jazz, soul* and *folk*. For each of the 10 genres we choose 5000 songs, giving a total of 50000 songs correctly tagged spreading over around 23000 transaction graphs. Other songs on an album that have been tagged as a genre outside of these 10 genres are included as features but are not processed individually as augmented subgraphs, since they do not belong to one of our evaluation genres. We bin our chosen numeric features, loudness and tempo, into 50 bins and extract (up to) the top 100 most frequent listeners for each song. These feature choices allows songs of which we have no listener data to be clustered on acoustic features only.

5.2 Cluster Quality Metric. The CORA dataset provides topic classifications of publications and is used in [21] as a means of providing ground truth for evaluation of cluster quality. We use the last.fm tags as the ground truth for the MSD graph.

Cluster quality is measured using the *cluster purity measure*[4, 21]. The purity of each cluster is calculated by taking the most frequent class in each cluster and determining the fraction of members of the cluster that belong to this class. For the MSD experiment this is straightforward as each song occurs only once in the stream and it is always associated with one genre. However, in the case of the CORA ground truth, we

[2] http://labrosa.ee.columbia.edu/millionsong/
[3] http://labrosa.ee.columbia.edu/millionsong/lastfm

must do some further preprocessing of the data as authors may occur multiple times in the stream, and not always with the same class label; authors may publish papers in various topics. Therefore, we use the paper classifications to create an association of the authors of each paper to the topic of their papers. Around 26% of authors had published a paper in more than one of the topic areas. In these cases we associate each author with the topic in which they most frequently published. 8.4% of authors had no single topic in which they published most frequently. In these cases, we associate all of the authors most frequent topics with the author. These classifications are used as a means of ground truth for our approach in this experiment.

Our approach is flexible in that it can result in both *hard* and *soft* clusters, as authors from the CORA graph may be placed in different clusters over the course of the stream resulting in a soft clustering, but in the MSD experiment each song will only be in one cluster and therefore a hard clustering. This affects the calculation of the cluster purity in the CORA graph. In our calculation of the cluster purity score, we only take into account authors in clusters where the majority of their membership belongs to that cluster. In the case where the highest membership of an author is shared with more than one cluster, the author is included in the purity calculations for all of those clusters.

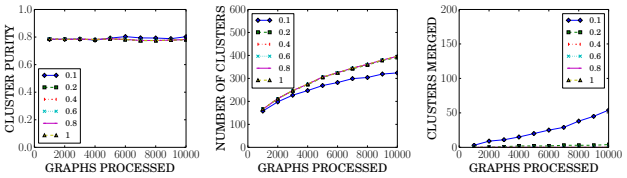
5.3 Results. Since our algorithm is novel, namely online vertex clustering, no other online vertex clustering methods exist to provide a suitable comparison. However existing offline algorithms can analyse these datasets for the same purpose, and we will do so to provide a baseline for comparison. An online streaming algorithm will encounter issues that the offline algorithms do not and we will be able to observe these differences in our experiments. The key difference is that the offline algorithm has the benefit of making multiple passes over the complete graph. This is in contrast to a streaming algorithm which will only be processing small sections of the graph at any time, and has limited access to the previous sections processed.

We note that we do not evaluate the efficiency of our approach but we expect it to be less efficient than *GSSClu* since for each transaction graph we cluster each vertex and its attributes (via the augmented subgraph) from the transaction graph individually.

5.3.1 Settings. Since we store the frequencies of each attribute type in a separate Count-Min sketch, we initialise each sketch with values chosen based on the estimated size of each feature space. The weight of each attribute is manually set to 1. The *HGS* size will be 25 and X, which determines when the merging algorithm is run, is set to 100.

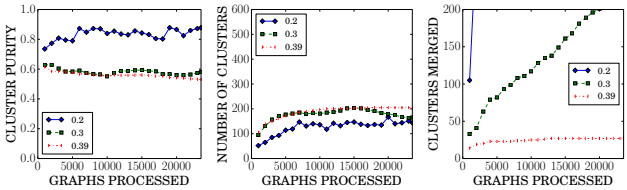
5.3.2 Effects of the Merging Algorithm Similarity Threshold on the Clustering. As our merging algorithm uses only the *HGS* structures from each cluster and we only consider the best values in each cluster, we can expect that even low thresholds can result in meaningful merges. We provide an overview of a range of similarity thresholds along with a finer grained analysis of thresholds we found to be particularly influential.

Figures 2a and 3a show the average purity versus the number of transaction graphs processed for a range of different similarity thresholds on the CORA and MSD graphs respectively. Figure 2a shows that



(a) Cluster Purity (b) Cluster Count (c) Merge Count

Figure 2: Effect of cluster merge threshold value on the CORA graph (*SoB* size of 200).



(a) Cluster Purity (b) Cluster Count (c) Merge Count

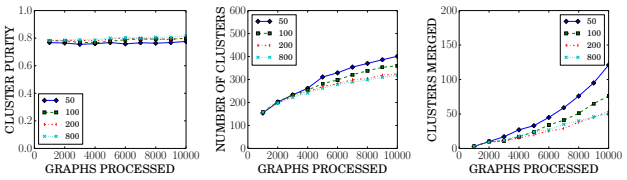
Figure 3: Effect of cluster merge threshold on the MSD graph (*SoB* size of 200).

for the CORA graph varying the threshold has little effect on the cluster purity which remains relatively constant. The results for the MSD graph in Figure 3a are somewhat different in that the cluster purity significantly decreases when the threshold is increased from 0.2 to 0.3. On closer investigation, clusters are of poor quality at 0.2, with one very large impure cluster of 49463 subgraphs. The reason for the higher purity is due to all of the other clusters at this threshold being small in size with a high purity. Thresholds of 0.3 resulted in a large cluster of 25173 subgraphs. For thresholds of 0.4 and above, too many clusters are created, with few merges, resulting in excessive computational times.

However with a threshold of 0.39 around 30 merges occur, which is enough to create distinct clusters early in the stream and prevent excessive clusters being created. This resulted in a much better distribution of subgraphs among clusters with the largest cluster being just 1403 subgraphs in size. Figures 2b and 3b show the number of clusters versus the number of graphs processed for a range of different similarity thresholds for the CORA and MSD graphs, respectively. For almost all thresholds on the CORA graph, the performance is similar, except for a threshold 0.1 where the increase in the number of clusters is significantly reduced. This result is reflected in Figure 2c which shows the number of merges versus the transaction graphs processed. Here the number of merges increased with 0.1 compared to the other thresholds. The reason for this is as the cosine similarity threshold decreases, more clusters will meet or exceed this threshold. Figures 3b and 3c show the same results for the MSD graph. In Figure 3b the general trend is that the number of clusters again increase with the number of transaction graphs processed. At a threshold of 0.3 and 0.39 the cluster numbers trend similarly, until 15000 graphs have been processed and the threshold of 0.3 leads to cluster numbers falling to a final number of clusters similar to 0.2, while 0.39 remains stable. A threshold of 0.39 is the optimal value as it maintains a better distribution of cluster sizes, which is qualitatively more useful in practice, and still achieves a good average purity. Once again these results are inversely reflected in Figure 3c, which shows that as the threshold is reduced the number of cluster merges increases. In summary, the general trends for the CORA dataset are clearer, in that the cluster purity and the number of clusters remain unaffected over most of the range of thresholds. The cluster purity remains more or less constant versus the number of transaction graphs processed, whilst the number of clusters increases. The situation for the MSD graph is more complicated with excessive clusters created at a threshold of 0.4 and above, to the extent that given our resources we had to stop the clustering at these thresholds early. Thresholds at 0.3 and below were too low, both allowing very large clusters to form. The reason for this is that many clusters had a cosine similarity of at least these values. This may be because the clusters in the MSD graph are naturally less distinct than those of the CORA graph.

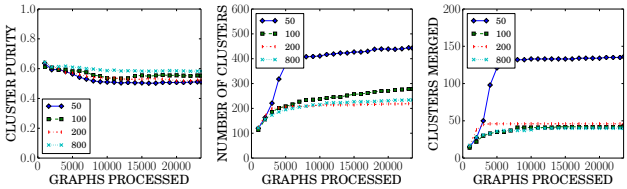
5.3.3 Effects of *SoB* Size on the Clustering. Now we will look at the effect of varying the *SoB* size. For each dataset we will choose a threshold for the merging algorithm that is optimal.

Figures 4a and 5a show the cluster purity versus graphs processed, with a range of *SoB* sizes, for the



(a) Cluster Purity (b) Cluster Count (c) Merge Count

Figure 4: The effect of varying the *SoB* parameter on the CORA graph (merge threshold 0.1).



(a) Cluster Purity (b) Cluster Count (c) Merge Count

Figure 5: The effect of varying the *SoB* parameter on the MSD graph (merge threshold 0.39).

CORA and MSD graphs respectively. It shows cluster purity remains more or less constant at values of 0.8 and between 0.5-0.6 for the CORA and MSD graphs respectively. Figures 4b and 5b show the number of clusters increases versus transaction graphs processed. This increase is greater in both with a decrease in *SoB* size. This result is replicated in Figure 4c and 5c for both graphs, which shows that the rate of increase in cluster merges is greater for reduced *SoB* sizes. The reason for this is that smaller *SoB* sizes will have a tighter mean due to a more limited number of the top substructure sample values in the *SoB*. One interesting observation is that in Figure 4c and 5c the number of merges reflect the trend seen when varying the threshold parameter for the merging algorithm in Figures 2c and 3c, except lower values result in more merges occurring for the same threshold.

5.3.4 Comparison with the Louvain method. Now that we have explored the influence of the parameters of our method, in this experiment we compare our streaming approach with a state-of-the-art non-streaming and offline Louvain method. We applied the Louvain method to the same augmented graphs but in batch, rather than a streaming mode. This achieved an average purity score of 0.86 over 118 clusters for the CORA graph, and 0.46 over 116 clusters for the MSD graph. Then, based on the CORA experiments in subsections 5.3.2 and 5.3.3, we selected values of 800

and 0.1 for the SOB size and cluster merging threshold respectively. With the chosen parameter settings for our approach, we achieved an average cluster purity of 0.82 over 318 clusters for the CORA graph, which is 0.04 lower than the Louvain method. These values were the best compromise between average cluster purity and number of clusters, but for all parameters we found similar average cluster purities. For the MSD graph with the optimal merge threshold of 0.39, all SoB sizes we achieved a higher average cluster purity than the Louvain method. The highest average cluster purity was 0.58 over 234 clusters with an SoB size of 800, 0.12 higher than the Louvain method.

6 Conclusions and Future Work

In this paper we present the first approach for vertex clustering in the augmented graph stream setting, and show potential applications via experimentation on two real world datasets from different application domains. We also show how the addition of vertex attributes into the clustering process aids the clustering algorithm by not only taking into account connectivity between the original structural vertices, but also extra information via a unified similarity measure. Furthermore, it can be difficult to define the number of clusters a pirori, particularly so for an unbounded stream and we show how our approach solves this challenge by dynamically creating new clusters as required. We show that our cluster merging step is able to maintain a high average cluster purity and low cluster count.

To conclude, in this work we presented a streaming algorithm that incorporates attributes directly into the vertex clustering process and demonstrated how its cluster purity is comparable or better than a well known offline approach but with the advantages of high scalability that comes with online streaming methods.

For future work we would like to look into extending our approach to very high dimensional data which would likely introduce significant efficiency challenges.

Acknowledgement

We would like to thank Dr. Michael Davis and Dr. Xin Cao for their useful feedback.

References

[1] P. K. Agarwal, G. Cormode, Z. Huang, J. Phillips, Z. Wei, and K. Yi. Mergeable summaries. *PODS '12*, pages 23–34.

[2] C. C. Aggarwal. On Classification of Graph Streams. *SDM '11*, pages 652–663.

[3] C. C. Aggarwal, Y. Li, P. S. Yu, and R. Jin. On dense pattern mining in graph streams. *PVLDB '10*, pages 975–984.

[4] C. C. Aggarwal, Y. Zhao, and P. S. Yu. On Clustering Graph Streams. *SDM '10*, pages 478–489.

[5] C. C. Aggarwal, Y. Zhao, and P. S. Yu. Outlier detection in graph streams. *ICDM '11*, pages 399–409.

[6] T. Bertin-Mahieux, D. P.W. Ellis, B. Whitman, and P. Lamere. The Million Song Dataset. *ISMIR '11*.

[7] V. Blondel and J. Guillaume. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008.

[8] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[9] S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3):75–174, 2010.

[10] A. K. Jain. Data clustering: 50 years beyond K-means. *Pattern Recognition Letters*, 31(8):651–666, June 2010.

[11] T. Kanungo, S. Member, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE TPAMI*, 24(7):881–892, 2002.

[12] B. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell system technical journal*, 49(2):291–307, 1970.

[13] J. Silva, E. Faria, and R. Barros. Data stream clustering: A survey. *ACM Computing Surveys (CSUR)*, 46:1–41, 2013.

[14] Y. Sun, J. Tang, J. Han, M. Gupta, and B. Zhao. Community Evolution Detection in Dynamic Heterogeneous Information Networks. *MLG '10*, pages 137–146.

[15] C. Tsourakakis and C. Gkantsidis. Fennel: Streaming graph partitioning for massive scale graphs. *ICWSDM '14*, pages 333–342.

[16] C. Wang, J. Lai, and P. Yu. Dynamic community detection in weighted graph streams. *SDM '13*, pages 151–161.

[17] G. Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. *Machine learning*, 23(1):69–101, 1996.

[18] Z. Xu, Y. Wang, and J. Cheng. A Model-based Approach to Attributed Graph Clustering. *SIGMOD '12*, pages 505–516.

[19] J. Yang, J. McAuley, and J. Leskovec. Community Detection in Networks with Node Attributes. *ICDM '13*, pages 1151–1156.

[20] M. Yuan, G. Jacques-silva, and Y. Lu. Efficient Processing of Streaming Graphs for Evolution-Aware, *CIKM '13*, pages 139–150.

[21] Y. Zhao and P. S. Yu. On Graph Stream Clustering with Side Information. *SDM '13*, pages 139–150.

[22] Y. Zhou, H. Cheng, and J. X. Yu. Graph Clustering Based on Structural / Attribute Similarities. *PVLDB '09*, 2(1):718–729.

[23] Y. Zhou, H. Cheng, and J. X. Yu. Clustering Large Attributed Graphs: An Efficient Incremental Approach. *ICDM '10*, pages 689–698.